Introduction to the Fortran 2003 Standard

Tom Clune SIVO
Hamid Oloso SIVO/AMTI
Code 610.3

Outline

- Series logistics
- Fortran resources
- Summary of new features
- Next session - Interoperability with C

Future Sessions

- Informal “brown-bag” format
  - SIVO can arrange for pizza if there is strong interest
- Focus on single feature (or small set of related features)
- Emphasis:
  - Feature availability in various compilers
  - Useful concrete examples
  - Pitfalls
  - Best practices
- Attendees encouraged to bring laptops to experiment with features
  - Room has “CNE” and “Guest-CNE” wireless
  - SIVO can assist with accessing NAG and IBM compilers online at NCCS and NAS respectively.

Tentative Schedule

- Roughly bi-weekly subject to other demands on speakers time
  - Would weekly be preferable?
- SIVO is interested in guest speakers - contact me if you are interested in hosting a session.

- Feb 12 - Interoperability with C
- Feb 26 - Allocatable/Pointer
- Mar 11 - I/O extensions
- Mar 25 - Miscellaneous A
- Apr 08 - Parameterized Types

- Apr 22 - Intro to OO
- May 06 - OO Inheritance
- May 20 - OO Polymorphism
- Jun 03 - Miscellaneous B

Fortran Resources

- Documents
  - "The New Features of Fortran 2003" by John Reid ftp://ftp.nag.co.uk/sc22wg5/N1601-N1650/N1648.pdf
  - Fortran 95/2003 Explained by M. Metcalf, J. Reid, and M. Cohen
  - *The* standard: http://www.open-std.org/jtc1/sc22/open/n3661.pdf
- The Modeling Guru: http://modelingguru.nasa.gov
  - Knowledge Base, discussion area (forum), and support
  - Most areas are publicly visible (requires registration to post)
    - All NCCS users have accounts by default (LDAP)
    - Almost anyone else can request and receive an account
  - Excellent place for community discussion of new Fortran features.
    - Look under "Languages, Libraries and Tools" at top level
    - Materials for this series will be in "Documents" for this section.
- Fortran mailing list: http://www.jiscmail.ac.uk/lists/comp-fortran-90.html

The Fortran Standard Process

- Starting with Fortran 90, a nominal schedule for future revisions was established:
  - Alternating major/minor revision each 5 years. (F95 was a minor release.)
  - Latest major release was F2003 (slipped 3 years)
  - Next "minor" release has slipped to ~2009.
    - Contains too many "major" feature requests including Co-Arrays.
- Plug for the standards committee: WG5
  - Work on standard is difficult, intensive, and ***essential***!
  - Process is public - anyone can submit feature requests
    - Advocate on committee is typically essential for successful incorporation
  - Scientific modeling is not well represented on the committee
    - Membership dominated (~50%) by industry (Fortran vendors)
    - Remainder of committee is odd mix of language theorists, private consultants, and a couple of individuals with a modeling interest.
  - Meets ~ 4 times per year. Usually in Las Vegas to save money.
  - Roughly 1 trip per year is joint with European committee

Compiler/Feature Availability

- At this time no vendor is providing a full implementation of the 2003 standard
  - *But* some vendors are *very* close.
  - As each feature is covered in later sessions, we will attempt to provide current status on implementation by major vendors.
  - Note that two TRs which were “promised” back at the introduction of F95 are widely implemented.
- Compilers with (known) major F2003 functionality:
  - IBM, NAG, (and we’ve been told Cray)
  - Note - need to use *latest* release
- Vendors with at least some F2003 features:
  - g95, gfortran, Intel, MIPSpro, …

Major Categories of Features

- Just the highlights today …
  - Emphasizing capabilities *not* syntax
  - Please be patient - we're still learning this too.
- Type 2 Technical Reports (TRs)
  - "promised" for F2003 at release of F95
  1. Allocatable dummy arguments and type components
  2. Support for IEEE floating point exceptions
- Interfacing with C
- Object orientation
- Lots of smaller enhancements
  - Most, but probably not all will be covered by this series.

Performance?

- Can new language feature *x* negatively impact performance?
  - Fears are fed by anecdotal evidence from colleagues and even gurus.
  - The answer of course is yes, ***but*** …
    - Impact greatly depends upon how and where used.
    - Programmer time matters too!
- Examples - in increasing order of paranoia
  - Object orientation incurs too much overhead
  - Scripted languages are too slow compared to compiled languages
    - But … libraries are often highly optimized
    - But … only use for "sewing" together top-level application
  - Procedure calls are "expensive"
    - But … convenience, development time, and *correctness* matter
    - But … overhead is trivial unless in innermost loop
  - Compiled languages are slow compared to assembly programming.
    - But … what about portability and development time?
  - Software is expensive compared to customized hardware …

Interoperability with C

- New syntax provides means both to call C procedures from Fortran *and* call Fortran procedures from C
  - Before, no standard conforming portable mechanism
  - Feature should expand the availability of libraries for the Fortran community - no more need for Fortran wrappers
  - Should also simplify computing environment - e.g. discover would not need a separate MPI build for each version of each Fortran compiler.
- Support largely derives from the ISO_C_BINDING intrinsic module
  - Named constants hording kind type parameter values for intrinsic types
    - Not all values are required to be standard conforming
  - Module procedures provide support for C addresses and pointers.
- BIND(C) attribute
  - Required for derived types, procedures, and global variables
  - Provides means for entity to have different names in each language

ALLOCATABLE Extensions

- Allocatable components and dummy arguments
  - Safer than POINTER components and dummy arguments
  - Potentially higher performance - unit stride and lack of aliasing
- New allocatable assignment syntax
  - Follows treatment of allocatable components in TR15581
  - If LHS is unallocated, it is allocated with shape of RHS
  - If LHS is different shape than RHS, it is *reallocated* to correct shape!
  - New intrinsic MOVE_ALLOC() moves an allocation from one variable to another (deallocating the original).

POINTER Extensions

- Pointer assignment
  - Can specify lower bounds. E.g. ptr(0:) => target
  - Can associate multidimensional pointers with rank-one arrays
    ptr(1:IM,1:JM,1:KM) => flatArray(1:IM*JM*KM)
    - RHS *must* be rank-1, but can be strided.
- INTENT attribute added for POINTER dummy variables
  - Refers to the association status *not* the target
  - Disallowed in F95 due to potential ambiguity.
- Procedure pointers
  - Permits “dynamic” binding where the behavior is not determined until run time.
    - Dummy procedure arguments are ultimately static procedure references higher in the calling tree.
  - May have explicit or implicit interfaces

Input/Output Enhancements

- Stream I/O
  - Analogous to C file access
  - Random access of ‘DIRECT’
  - Flexible record lengths of ‘SEQUENTIAL’
- Asynchronous I/O for performance
  - Opened with ASYNCHRONOUS=‘YES’
  - Other operations may proceed while I/O statement is in execution
  - Mechanisms provided to test/ensure completion of operations.
  - *Performance impact is a quality-of-implementation issue.*
- Recursive I/O for internal files
- Miscellaneous
  - User specified error messages for I/O statements
  - Edit descriptors for derived types
  - FLUSH, ROUND, DECIMAL, SIGN, NEW_LINE
  - IEEE exceptional values

Access to the OS

- Until now, the Fortran language has intentionally avoided admitting the existence of such things as operating systems and even compilers.
  - Desire to not preclude unforeseen implementations
  - Painful constraint for practical programming
- Extensions
  - New intrinsic procedures to
    - Access the command line arguments
    - Environment variables
  - Intrinsic module for I/O parameters: ISO_FORTRAN_ENV
    - Standardizes values that were previously implementation dependent.
    - Units for input, output, and error reporting
    - IOSTAT_END and IOSTAT_EOR
    - Storage sizes for numeric, character, and files.

Object Orientation

- Object orientation (OO) is a very powerful programming paradigm that is common to almost all modern programming languages.
  - Basic concept: strongly couple data structures with the procedures that access/modify them.
    - Such coupled entities are known as ***classes***.
    - Individual instances of such data structures are referred to as ***objects***.
    - Procedures that operate on objects are referred to as ***methods***.
  - Associated concepts:
    - *Encapsulation* allows the internal details of a class to be hidden from other parts of a program. This tends to avoid dependencies and enable isolated changes to software.
    - *Inheritance* allows new classes (children or subclasses) to be defined in terms of extensions to other classes (parents or superclasses).
    - *Polymorphism* allows variables to be associated with any members of an inheritance tree. Such abstraction can be very powerful.
- The relevance of OO to purely numerical algorithms is marginal. However, most complex models contain significant amounts of "infrastructure" which is often very amenable to OO techniques.
- This series will have a separate session discussing OO from a modelers vantage point prior to examining F2003 features which support OO.

Type-bound Procedures

- In F2003, methods are attached to data structure via type-bound procedures.
  - Derived type declaration allows for a CONTAINS clause to declare these methods.
  - Methods can be PRIVATE or PUBLIC - specified individually, in aggregate or default to public.
  - Includes support for overloading of methods (including operators)
  - By default the first argument to a method is passed implicitly via a OO style of invocation:

    call obj % foo(x) ! Passes x *and* obj to foo
    - Can specify different argument to attach to object, or none.

Type Extension (OO)

- Fortran 2003 supports *single* inheritance by declaring that a new derived type EXTENDS a previously defined type.
  - New type includes all components of the parent type.
  - New type includes all methods of the parent type.
- Important concept: A type is said to be ***compatible*** with another type if it is the same type or an extension of that type.
- Procedures can be declared to be NON_OVERRIDABLE
- Extended types have a special parent component
  - Type and type parameters are those of the parent type
  - Name is the name of the parent type

Polymorphic Entities

- In F2003, the CLASS keyword is used to indicate a polymorphic entity.
  - Must have POINTER or ALLOCATABLE attribute or be a dummy argument
  - Dynamic (run-time) type is correspondingly obtained from allocation, pointer assignment, or argument association.
- Objects can be declared as “unlimited polymorphic”
  - Type is different than all other entities
  - Type is compatible with all entities
- F2003 has inquiry functions to determine if two entities are of the same dynamic type or if one’s dynamic type is an extension of another
- F2003 provides a mechanism to conditionally execute different blocks of code depending on the dynamic type of an object.
  - SELECT TYPE syntax is similar to SELECT CASE from F90.

Abstract Types

- Developers often encounter a situation where there is no appropriate default implementation for a method in a base class.
  - In OO languages, such classes are call *abstract.*
  - Non-abstract classes are referred to as *concrete*.
- F2003 supports abstract types which may have DEFERRED implementations for the methods.
  - Each deferred implementation requires an explicit ABSTRACT interface.
  - Deferred bindings are only permitted in an abstract type.
- Only polymorphic entities may be declared with an abstract type.
  - Not permissible to declare, allocate, or construct a non-polymorphic object of abstract type.

Parameterized Types

- Flexible facility that allows *type* and *kind* values for user defined types:
  - KIND parameters
    - Constant (compile time parameters)
    - Can be used as the kind declaration for components
  - LENGTH parameters
    - Dynamic (variable at run time)
    - Modeled after lengths of character components and array bounds.
- Supports keyword syntax: kind=…,len=…
- Default values may be given in an initialization expression.
- Can be used to overload interfaces

Miscellaneous

- Modules
  - Intrinsic modules
  - IMPORT statement - allows INTERFACE bodies to access host module.
- C-style enumerations
- ASSOCIATE construct - convenient where identical expressions exist multiple times within a block of code.
- VOLATILE attribute - warns the compiler that value may be changed by some other aspect of the system.
- '[',']' far array constructors as alternative to '(/',''/)'
- Names can be up to 63 characters (up from 31)
- Up to 256 continuation lines (up from 39)
- All intrinsics permitted in specification and initialization
- Independent private/public attribute for type components

Next Session

- Interoperability with C
  - Brown-bag format - bring your lunch!
  - Tuesday February 12 @ 12:00
  - B28-E210
  - Hopefully will have webex sorted out by then.